The Trellis Complexity of Convolutional Codes

Robert J. McEliece, *Fellow, IEEE*, and Wei Lin, *Student Member, IEEE*

***Abstract*— It has long been known that convolutional codes have a natural, regular, trellis structure that facilitates the implementation of Viterbi's algorithm [35], [11]. It has gradually become apparent that linear block codes also have a natural, though not in general a regular, "minimal" trellis structure, which allows them to be decoded with a Viterbi-like algorithm [2], [36], [25], [12], [30], [16], [13], [18], [27], [28], [9], [17]. In both cases, the complexity of an unenhanced Viterbi decoding algorithm can be accurately estimated by *the number of trellis edge symbols per encoded bit*. It would therefore appear that we are in a good position to make a fair comparison of the Viterbi decoding complexity of block and convolutional codes. Unfortunately, however, this comparison is somewhat muddled by the fact that some convolutional codes, the *punctured* convolutional codes [5], are known to have trellis representations which are significantly less complex than the conventional trellis. In other words, the conventional trellis representation for a convolutional code may not be the "minimal" trellis representation. Thus ironically, we seem to know more about the minimal trellis representation for block than for convolutional codes. In this paper we provide a remedy, by developing a theory of minimal trellises for convolutional codes. (A similar theory has recently been given by Sidorenko and Zyablov [32].) This allows us to make a direct performance–complexity comparison for block and convolutional codes. A by-product of our work is an algorithm for choosing, from among all generator matrices for a given convolutional code, what we call a *trellis-canonical* generator matrix, from which the minimal trellis for the code can be directly constructed. Another by-product is that in the new theory, punctured convolutional codes no longer appear as a special class, but simply as high-rate convolutional codes whose trellis complexity is unexpectedly small.**

***Index Terms*— Convolutional code, minimal trellis, decoding complexity.**

I. Introduction

WE BEGIN with the standard definition of a convolutional code [10], [29], always assuming that the underlying field is $F = \mathrm{GF}(2)$. An (n, k) convolutional code $\mathcal{C}$ is a k-dimensional subspace of $F(D)^n$, where $F(D)$ is the field of rational functions in the indeterminate D over the field F. The *memory*, or *degree*, of $\mathcal{C}$, is the smallest integer m such that $\mathcal{C}$ has an encoder requiring only m delay units. An (n, k) convolutional code with memory m is said to be an (n, k, m) convolutional code. The *free distance* of $\mathcal{C}$ is the minimum Hamming weight of any codeword in $\mathcal{C}$. An (n, k, m) convolutional code with free distance d is said to be an (n, k, m, d) code.

A *canonical generator matrix*[1] $G(D)$ for an (n, k, m) convolutional code $\mathcal{C}$ is a $k \times n$ matrix with polynomial entries, whose row space is $\mathcal{C}$, such that the direct-form realization of an encoder for $\mathcal{C}$ based on $G(D)$ uses exactly m delay elements [10], [29]. From a canonical generator matrix $G(D)$, or rather from a physical encoder built using $G(D)$ as a blueprint, it is possible to construct a "conventional" trellis representation for $\mathcal{C}$. This trellis is in principle infinite, but it has a very regular structure, consisting (after a short initial transient) of repeated copies of what we shall call the *trellis module* associated with $G(D)$. The trellis module consists of 2^m "initial states" and 2^m "final states," with each initial state being connected by a directed edge to exactly 2^k final states. Thus the trellis module has 2^{k+m} edges. Each edge is labeled with an n-symbol binary vector, namely, the output produced by the encoder in response to the given state transition. Thus each edge has length (measured in coded "symbols") n, and so the total edge length of the conventional trellis module is $n2^{k+m}$. Since each trellis module represents the encoder's response to k input bits, we are led to define the "conventional trellis complexity" of the trellis module as

$$\frac{n}{k} \cdot 2^{m+k} \quad \text{symbols per encoded bit} \tag{1.1}$$

or *symbols per bit*, for short. If the code $\mathcal{C}$ is decoded using Viterbi's maximum-likelihood algorithm on the trellis [35], [11], the work factor involved in updating the metrics and survivors at each trellis module is proportional to the edge length of the trellis module, so that the trellis complexity as defined in (1.1) is a measure of the effort *per decoded bit* required by Viterbi's algorithm. (For a more detailed discussion of the complexity of Viterbi's algorithm on a trellis, see [28, sec. 2].)

For example, consider the $(3, 2, 2)$ convolutional code with canonical generator matrix given by

$$G_1(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{pmatrix}. \tag{1.2}$$

This code has the largest possible free distance, viz., $d_{\text{free}} = 3$, for any $(3, 2, 2)$ code. A "direct-form" encoder based on the generator matrix $G_1(D)$ is shown in Fig. 1. If the input pair is (u_1, u_2) and the state of the encoder is (s, t), then the output

Manuscript received December 15, 1995; revised May 12, 1996. This work was supported in part by NSF under Grant NCR-9505975 and under a Grant from Pacific Bell. A portion of the work was also performed at the Jet Propulsion Laboratory, California Institute of Technology, under Contract to the National Aeronautics and Space Administration. The material in this paper was presented at the 3rd International Symposium on Communication Theory and Exposition, Ambleside, UK, July 1995.

The authors are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA.

Publisher Item Identifier S 0018-9448(96)07301-4.

[1] Originally called *minimal* by Forney [10]. However, following a recent suggestion by Forney, Johannesson, and Wan [10], we shall use the term *canonical*, which is also adopted in [29].

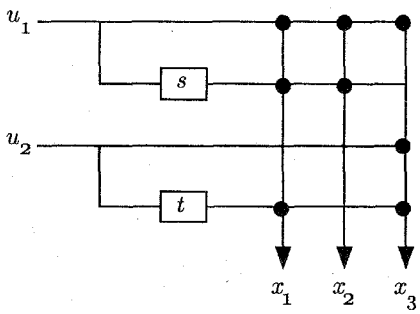

Fig. 1. A direct-form encoder based on the generator matrix $G_1(D)$ in (1.2). The input is (u_1, u_2), the output is (x_1, x_2, x_3), and the state of the encoder is (s, t). (The boxes labeled s and t are unit delay elements.)

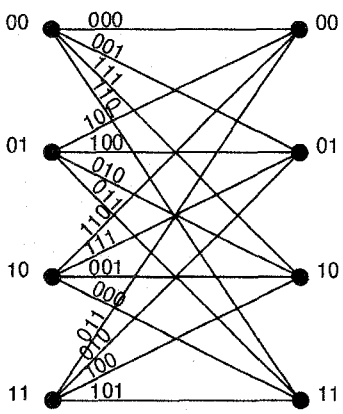

Fig. 2. The conventional trellis module for the code with canonical generator matrix $G_1(D)$ given in (1.2).

(x_1, x_2, x_3) is given by

$$\begin{aligned} x_1 &= u_1 + s + t \\ x_2 &= u_1 + s \\ x_3 &= u_1 + u_2 + t \end{aligned} \tag{1.3}$$

and the "next state" is just the input pair (u_1, u_2). The conventional trellis module for the code with canonical generator matrix $G_1(D)$ given in (1.2) is shown in Fig. 2. The three-symbol edge label on the edge from (s, t) to (u_1, u_2) is the triple (x_1, x_2, x_3) given in (1.3). The total number of edge symbols is 48, so that the conventional trellis complexity corresponding to the matrix $G_1(D)$ is $48/2 = 24$ symbols per bit, as given in (1.1).

But we can do substantially better than this, if we use the fact that this particular code is a *punctured* convolutional code. We now briefly review the theory of punctured convolutional codes, to see how simplified trellises result.

If we begin with a "parent" $(N, 1, m)$ convolutional code, and "block" it to depth k, i.e., group the input bit stream into blocks of k bits each, the result is an (Nk, k, m) convolutional code. If we now delete, or "puncture," all but n symbols from each Nk-symbol output block, the result is an (n, k, m) convolutional code.[2] This punctured code can be represented by a trellis whose trellis module is built from k copies of the

[2]In fact, the memory of the punctured code may be less than m, but for most interesting punctured codes no memory reduction will take place.

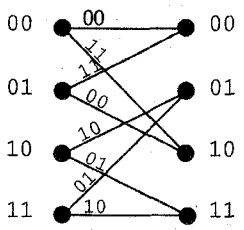

Fig. 3. The trellis module for the $(2, 1, 2)$ code with generator matrix $G_2(D) = (1 + D + D^2 \quad 1 + D^2)$. The total number of edge symbols is 16, and so the trellis complexity is 16 symbols per bit.

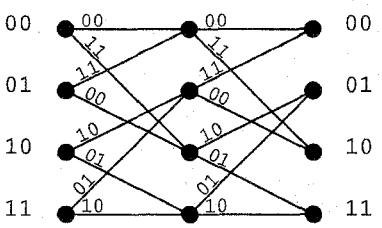

Fig. 4. The trellis module for the $(4, 2, 2)$ code obtained from the code of Fig. 3 by "blocking" the inputs in blocks of size 2. The total number of symbols is 32, and so the trellis complexity is $32/2 = 16$ symbols per bit, the same as for the original code.

trellis modules from the parent $(N, 1, m)$ code, each of which has only 2^{m+1} symbols, so that the total number of symbols on the trellis module is $n \cdot 2^{m+1}$, which means that the trellis complexity of an (n, k, m) punctured code is

$$\frac{n}{k} \cdot 2^{m+1} \quad \text{symbols per bit} \tag{1.4}$$

which is a factor of 2^{k-1} smaller than the complexity of the conventional trellis given in (1.1). For $k = 1$ this is no improvement, but for larger values of k the trellis complexity reduction afforded by puncturing becomes increasingly significant. And while the class of punctured convolutional codes is considerably smaller than the class of unrestricted convolutional codes, nevertheless many punctured convolutional codes with good performance properties are known [5], [15], [3], [8]. Punctured convolutional codes, especially high-rate ones, are often preferred in practice, precisely because of their reduced trellis complexity.

For example, consider the $(2, 1, 2, 5)$ convolutional code defined by the canonical generator matrix

$$G_2(D) = (1 + D + D^2 \quad 1 + D^2). \tag{1.5}$$

The conventional trellis module for this code is shown in Fig. 3.

If we "block" this code into blocks of size $k = 2$, we obtain a $(4, 2, 2)$ convolutional code, still with $d_{\text{free}} = 5$, for which the conventional trellis module is two copies of the trellis module shown in Fig. 3; see Fig. 4.

Now we can do the puncturing. Take the $(4, 2, 2)$ code, as represented by the trellis module in Fig. 4, and delete the second output symbol on each of the edges in the second part of the module. The result is shown in Fig. 5. This structure can be thought of as the trellis module for a $(3, 2, 2)$ code; the corresponding d_{free} turns out to be 3. According to

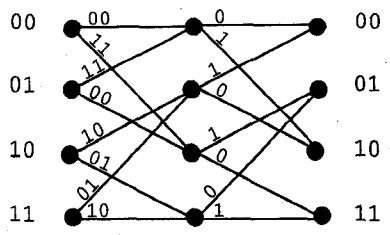

Fig. 5. The trellis module for the $(3,2,2)$ "punctured" code obtained from the code of Fig. 4 by deleting every fourth output symbol. The total number of edge symbols is $16+8=24$, and so the trellis complexity is $24/2=12$ symbols per bit.

(1.1), the conventional trellis complexity of a $(3,2,2)$ code is $3/2 \cdot 2^4 = 24$ symbols per bit. But if we use instead the "punctured" trellis corresponding to the $k=2$ blocked version of the parent $(2,1,2)$ code, we find from (1.4), or Fig. 5, that the trellis complexity is instead only $3/2 \cdot 2^3 = 12$ symbols per bit. In fact, it can be shown that this punctured $(3,2,2)$ code is the same as the "conventional" code with generator matrix $G_1(D)$ given in (1.2). (Indeed, this example is taken almost verbatim from [5], where it was used to illustrate the way puncturing can reduce decoding complexity.)

It seems mysterious that an ordinary-looking generator matrix like $G_1(D)$ produces a code whose trellis complexity can be significantly reduced (if one knows that it is, in fact, a punctured code), whereas for an almost identical code, namely, the one defined by the generator matrix

$$\begin{pmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{pmatrix}$$

no such reduction is possible. In Section II we will resolve this mystery by developing a simple algorithm for constructing the "minimum" trellis for any convolutional code.[3] Since a "punctured" trellis is simply one of many possible trellis representations of a given code, our technique will always find a trellis, with complexity at least as small as given by (1.4), even if we are not told in advance that the code can be obtained by "puncturing." But more important, it will often result in considerable simplification of the trellis representation of a convolutional code which is not a punctured code. We will illustrate this with worked examples in Sections II and III, and numerical tables in Section IV.

II. Construction of Minimal Trellises

If $G(D)$ is a canonical generator matrix for an (n,k,m) convolutional code C, then we can write $G(D)$ in the form

$$G(D) = G_0 + G_1 D + \cdots + G_L D^L \tag{2.1}$$

[3] In this paper, we have chosen the number of trellis symbols per encoded bit as our measure of trellis complexity. However, as we shall see in Section II, our construction is based on the well-developed theory of minimal trellises for block codes, and it is known that the minimal trellis for a block code simultaneously minimizes not only the total edge symbol count, but also the total vertex count, the total number of bifurcations, the maximum vertex dimension, and a number of other quantities [28], [34]. It follows that if one is interested in minimizing any of the analogous quantities for a convolutional code, the minimal trellis as defined here is the unique structure for doing the job.

where $G_0, \cdots, G_L$ are $k \times n$ scalar matrices (i.e., matrices whose entries are from GF(2)), and L is the maximum degree of any entry of $G(D)$. The integer L is called the *memory* of the code. If we concatenate the $L+1$ matrices $G_0, \cdots, G_L$, we obtain a $k \times (L+1)n$ scalar matrix, which we denote by $\tilde{G}$

$$\tilde{G} = (G_0 \quad G_1 \quad \cdots \quad G_L). \tag{2.2}$$

It is well known [26, ch. 9] that the matrix $\tilde{G}$ and its shifts can be used to build a "scalar" generator matrix G_{scalar} for the code C (for simplicity of notation we illustrate the case $L=2$)

$$G_{\text{scalar}} = \begin{bmatrix} G_0 & G_1 & G_2 & & & \\ & G_0 & G_1 & G_2 & & \\ & & G_0 & G_1 & G_2 & \\ & & & G_0 & G_1 & G_2 \\ & & & & \ddots & \end{bmatrix}. \tag{2.3}$$

The matrix in (2.3) is, except for the fact that it continues forever, the generator matrix for a binary block code (with a very regular structure), and so any of the techniques which have been developed for finding minimal trellises for block codes are useful for constructing trellis representations for convolutional codes. In the remainder of this section, we will adapt the techniques developed in [28, sec. 7], which show how to construct a trellis directly from any generator matrix for a given block code, and the *minimal trellis* if the generator is in "minimal-span" form, to construct a trellis for C based on the infinite scalar generator matrix G_{scalar}.

The trellis module for the trellis associated with G_{scalar} corresponds to the the $(L+1)k \times n$ "matrix module"

$$\hat{G} = \begin{pmatrix} G_L \\ G_{L-1} \\ \vdots \\ G_0 \end{pmatrix} \tag{2.4}$$

which repeatedly appears as a vertical "slice" in G_{scalar}. Using the techniques in [28, sec. 7], it is easy to show that the number of edge symbols in this trellis module is

$$\text{edge symbol count} = \sum_{j=1}^{n} 2^{a_j} \tag{2.5}$$

where a_j is the number of "active" entries in the jth column of the matrix module $\hat{G}$. (An element is called active if it belongs to the "active span" of one of the rows of $\tilde{G}$. We will elaborate on this below.) Our object is to find a generator matrix for which the edge symbol count in the corresponding trellis module is as small as possible.

To clarify these ideas, we consider the $(3,2,1)$ code with (canonical) generator matrix

$$G_3(D) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1+D & 1+D \end{pmatrix}. \tag{2.6}$$

According to (1.1), the conventional trellis complexity for this code is 12 symbols per bit. However, we can do better. The scalar matrix $\tilde{G}_3$ corresponding to $G_3(D)$ is (cf. (2.2))

$$\tilde{G}_3 = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix}. \tag{2.7}$$

In (2.7), we have shown the "active elements" of each row, i.e., the entries from the first nonzero entry to the last nonzero entry, in boldface. The *spanlength* of, i.e., the number of active entries in, the first row is therefore three; and the spanlength of the second row is six. The "matrix module" corresponding to $\tilde{G}_3$ is (cf. (2.4))

$$\hat{G}_3 = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 1 & 1 & 1 \end{pmatrix}.$$

Thus $a_1 = 3$, $a_2 = 3$,, and $a_3 = 3$, which by (2.5) means that the corresponding trellis module has $2^3 + 2^3 + 2^3 = 24$ edge symbols. Since each trellis module represents two encoded bits, the resulting trellis complexity is $24/2 = 12$ symbols per bit. Since we have already noted that the conventional trellis complexity for this code is also 12 symbols per bit, the trellis corresponding to $G_3(D)$ is not better than (in fact, it is isomorphic to) the conventional trellis. To do better, we need to find a generator matrix for the code for which $\Sigma_i 2^{a_i}$ is less than 24. Using the results of [28, sec. 6], it is possible to show that minimizing $\Sigma_i 2^{a_i}$ is equivalent to minimizing $\Sigma_i a_i$, i.e., the total spanlength of the corresponding $\tilde{G}$, and so we shall look for generator matrices for which the span of $\tilde{G}$ is reduced.

If we add the first row of $G_3(D)$ to the second row, the resulting generator matrix, which is still canonical, is

$$G_3'(D) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1+D & D \end{pmatrix}.$$

The scalar matrix $\tilde{G}_3'$ corresponding to $G_3'(D)$ is (cf. (2.2))

$$\tilde{G}_3' = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{pmatrix}. \tag{2.8}$$

The spanlenth of the first row of $G_3'(D)$ is three, and the spanlength of the second row is five, and so the total spanlength is eight, one less than that of $G_3(D)$. The "matrix module" corresponding to $\tilde{G}_3'$ is (cf. (2.4))

$$\hat{G}_3' = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 0 \end{pmatrix}.$$

Here $a_1 = 2$, $a_2 = 3$, and $a_3 = 3$, and so by (2.5), the corresponding trellis module has $2^2 + 2^3 + 2^3 = 20$ edge symbols, so that the resulting trellis complexity is $20/2 = 10$ symbols per bit.

But we can do still better. If we multiply the first row of $G_3'(D)$ by D and add it to the second row, the resulting generator matrix, which is still canonical, is

$$G_3''(D) = \begin{pmatrix} 1 & 0 & 1 \\ D & 1+D & 0 \end{pmatrix}.$$

The scalar matrix $\tilde{G}_3''$ corresponding to $G_3''(D)$ is (cf. (2.2))

$$\tilde{G}_3'' = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{pmatrix}. \tag{2.9}$$

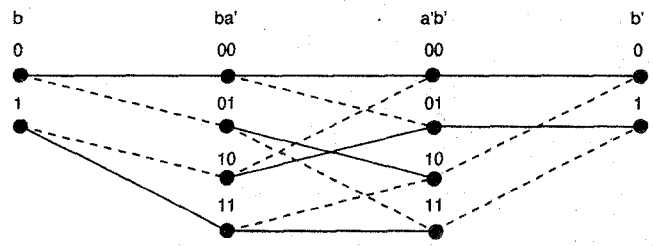

Fig. 6. The trellis module for the $(3, 2, 1)$ code with generator matrix $G_3''(D)$. This is the minimal trellis module for this code.

The spanlenth of $G_3''(D)$ is seven, one less than that of $G_3'(D)$. The "matrix module" corresponding to $\tilde{G}_3''$ is (cf. (2.4))

$$\hat{G}_3'' = \begin{matrix} a \\ b \\ a' \\ b' \end{matrix} \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 0 \end{pmatrix}. \tag{2.10}$$

Here $a_1 = 2$, $a_2 = 3$, and $a_3 = 2$, and so by (2.5), the corresponding trellis module has $2^2 + 2^3 + 2^2 = 16$ edge symbols, so that the resulting trellis complexity is $16/2 = 8$ symbols per bit. The trellis module itself, again constructed using the techniques described in [28, sec. 7] is shown in Fig. 6. The vertex labels in Fig. 6 represent the information bits corresponding to the rows of $\hat{G}_3''$.

(Note that in this example, the ratio of the conventional trellis complexity to the minimal trellis complexity is $12/8 = 3/2$. If this code were punctured, then according to (1.1) and (1.4), the ratio would be at least 2. Thus we conclude that the code with generator matrix $G_3(D)$ as given in (2.6) is not a punctured code, which shows that the theory of minimal trellises for convolutional codes goes beyond merely "explaining" punctured codes.)

Furthermore, it is easy to see that there is no generator matrix for this code with spanlength less than seven, so that the trellis module shown in Fig. 6 yields the minimal trellis for the code. Alternatively, we can examine the scalar generator matrix for the code corresponding to $\tilde{G}_3''$ (cf. (2.3))

$$G_{\text{scalar}} = \begin{bmatrix} \underline{1} & 0 & \overline{\underline{1}} & 0 & 0 & 0 & & & & & & & \\ 0 & \underline{1} & 0 & 1 & \overline{\underline{1}} & 0 & & & & & & & \\ & & & \underline{1} & 0 & \overline{\underline{1}} & 0 & 0 & 0 & & & & \\ & & & 0 & \underline{1} & 0 & 1 & \overline{\underline{1}} & 0 & & & & \\ & & & & & & 0 & \underline{1} & 0 & \overline{\underline{1}} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \overline{\underline{1}} & 0 & \\ & & & & & & & & & & \ddots & & \end{bmatrix}. \tag{2.11}$$

In (2.11), we see that G_{scalar} has the property that no column contains more than one underlined entry (the *L*eftmost nonzero entry in its row), or more than one overlined entry (the *R*ightmost nonzero entry in its row). Thus G_{scalar} has the "LR" property, and so, *if it were a finite matrix*, it would produce the minimal trellis for the code [28, sec. 6]. To circumvent the problem that G_{scalar} is infinite, we can define the *Mth truncation of the code* C, denoted by $C^{[M]}$, as the

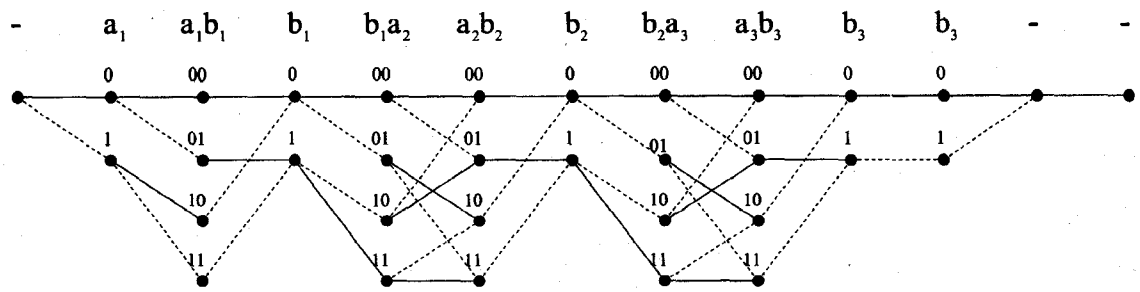

Fig. 7. The minimal trellis for the $[12, 6]$ block code obtained from the $M = 3$ truncation of the $(3, 2, 1)$ code with trellis-minimal generator matrix $G_3''(D)$. It is built from two copies of the minimal trellis module of Fig. 6, together with initial and final "transient" sections.

$[(M + L)n, Mk]$ block code obtained by taking only the first Mk rows of G_{scalar}, i.e., the code with $Mk \times (M + L)n$ generator matrix

$$G_{\text{scalar}}^{[M]} = \begin{bmatrix} \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & & & & \\ 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & & & & \\ & & & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 & & & \\ & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 & & & \\ & & & & & & & & & \ddots & & \\ & & & & & & \underline{1} & 0 & \overline{1} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \overline{1} & 0 \end{bmatrix}. \tag{2.12}$$

Plainly, if G_{scalar} has the LR property, so does $G_{\text{scalar}}^{[M]}$, for all $M \geq 1$. Thus it follows from the "standard" theory of trellises for block codes, that the matrix $G_{\text{scalar}}^{[M]}$ produces the minimal trellis for $\mathcal{C}^{[M]}$, for all M, and so we can safely call the infinite trellis, built from trellis modules corresponding to $\hat{G}$, *the* minimal trellis for the code.

For example, consider the $M = 3$ truncation, with corresponding scalar matrix

$$G_{\text{scalar}}^{[3]} = \begin{matrix} a_1 \\ b_1 \\ a_2 \\ b_2 \\ a_3 \\ b_3 \end{matrix} \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & & & & & & \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & & & & & & \\ & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 & & & \\ & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 & & & \\ & & & & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & 0 & 0 & 0 \\ & & & & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & 0 & 0 \end{pmatrix}$$

This is the generator matrix for a $[12, 6]$ block code. If we apply the method of [28, sec. 7] to $G^{[3]}$, we obtain the trellis shown in Fig. 7. As is seen, this trellis consists of two copies of the minimal trellis module of Fig. 6 "glued together," plus initial and final "transient" sections. Since $G^{[3]}$ is in "LR" from, we are guaranteed that the trellis of Fig. 7 is the minimal trellis for the code.

The preceding argument, though it was presented in terms of a specific example, is entirely general. It shows that a canonical (or simply basic) generator matrix $G(D)$ produces a minimal trellis if and only if $G(D)$ has the property that the spanlength of the corresponding $\tilde{G}$ cannot be reduced by an operation of the form

$$g_i(D) \leftarrow g_i(D) + D^{\ell} g_j(D)$$

where $g_i(D)$ is the ith row of $G(D)$, and ℓ is an integer in the range $0 \leq \ell \leq L$. We call a generator matrix with this property a *trellis-canonical* generator matrix for $\mathcal{C}$. A trellis-canonical generator matrix must be canonical, but the converse need not be true, as the example of this section shows. The set of trellis-canonical generator matrices for a given code $\mathcal{C}$ coincides with the set of generator matrices for which the spanlength of the corresponding $\tilde{G}$ is a minimum. In the next section, we will give two more examples of minimal trellises.

III. Two More Examples

Our first example is for the code whose generator matrix is given in (1.2). The corresponding decomposition (cf. (2.1)), is

$$G_1(D) = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} D.$$

The scalar matrix $\tilde{G}$ is thus

$$\tilde{G}_1 = \begin{pmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \end{pmatrix}.$$

It is easy to see that the corresponding "G_{scalar}" has the LR property, so $G_1(D)$ is trellis-canonical. The "matrix module" $\hat{G}$ obtained from (2.4) is

$$\hat{G}_1 = \begin{matrix} a \\ b \\ a' \\ b' \end{matrix} \begin{pmatrix} \mathbf{1} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & \mathbf{1} \end{pmatrix}. \tag{3.1}$$

Since there are three active entries in each column of $\hat{G}$, it follows from (2.5) that the edge symbol count for the trellis module is $2^3 + 2^3 + 2^3 = 24$, so that the trellis complexity for this trellis module is $24/2 = 12$ symbols per bit, the same as given by (1.4) for the punctured trellis. To actually construct the trellis module, we can use the techniques of [28, sec. 7], and the result is shown in Fig. 8. (This code is the first code listed in Table III in the Appendix.)

As our second example, we consider a "partial-unit-memory" code, taken from [22], [1]. It is an $(8, 4, 3)$ code with $d_{\text{free}} = 8$, and with canonical generator matrix (as taken from [1])

$$G(D) = \begin{pmatrix} 11111111 \\ 11101000 \\ 10110100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10101100 \\ 10010110 \end{pmatrix} D. \tag{3.2}$$

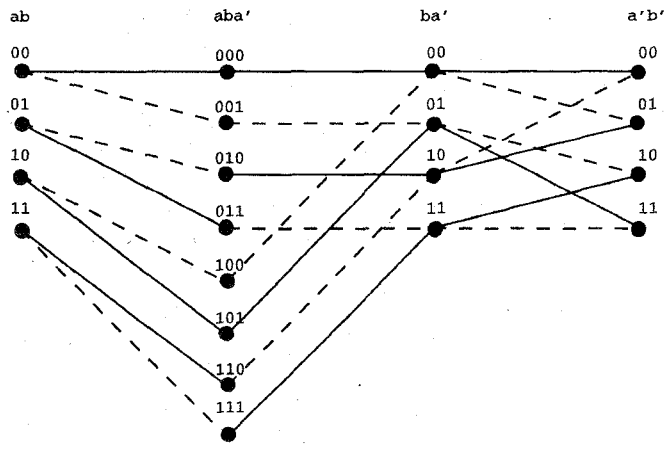

Fig. 8. The trellis module for the $(3,2,2)$ code with generator matrix $G_1(D)$. This module is isomorphic to the one in Fig. 5.

The conventional trellis complexity for this code is by (1.1) $8/4 \cdot 2^7 = 256$ symbols per bit. We can reduce this number to 120, as follows. First we concatenate the two matrices in (3.2), obtaining the following 4×16 scalar matrix $\tilde{G}$:

$$\tilde{G} = \begin{pmatrix} 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 1&1&1&0&1&0&0&0&1&1&0&1&1&0&0&0 \\ 1&0&1&1&0&1&0&0&1&0&1&0&1&1&0&0 \\ 1&0&0&1&1&0&1&0&1&0&0&1&0&1&1&0 \end{pmatrix}.$$

Next, using the techniques developed in [28, sec. 6], we perform a series of elementary row operations on $\tilde{G}$, transforming it to the "minimal span," or "trellis oriented" form $\tilde{G}'$.

$$\tilde{G}' = \begin{pmatrix} \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&0&0&0&0&0&0&0&0 \\ 0&0&0&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&0&0&0 \\ 0&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&0&0 \\ 0&0&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&0 \end{pmatrix}. \tag{3.3}$$

The "matrix module" $\hat{G}$ defined in (2.4) is thus

$$\hat{G} = \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ \mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&0&0&0 \\ \mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&0&0 \\ \mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0}&\mathbf{1}&0 \\ \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1} \\ 0&0&0&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1} \\ 0&\mathbf{1}&\mathbf{0}&\mathbf{0}&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1} \\ 0&0&\mathbf{1}&\mathbf{0}&\mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{0} \end{pmatrix}$$

and so by (2.5) that the total number of edge symbols in the trellis module is $2^4+2^5+2^6+2^7+2^7+2^6+2^5+2^4 = 480$. Since each trellis module represents four encoded bits, it follows that the trellis complexity is $480/4 = 120$ symbols per bit, compared to the conventional trellis complexity, cited above, of 256 symbols per bit.

The matrix G_{scalar} corresponding to the matrix $\tilde{G}'$ in (3.3) is easily seen to have the LR property, and so the generator matrix (cf. (3.3))

$$G'(D) = \begin{pmatrix} 1&1&1&1&1&1&1&1 \\ 0&0&0&1&0&1&1&1 \\ 0&1&0&0&1&0&1&1 \\ 0&0&1&0&1&1&1&0 \end{pmatrix}$$

$$+ \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ 1&1&0&1&1&0&0&0 \\ 1&0&1&0&1&1&0&0 \\ 0&0&1&1&1&0&1&0 \end{pmatrix} D$$

is trellis-canonical. However, the trellis complexity can be reduced still further, if we allow *column permutations* of the original generator matrix $G(D)$ in (3.2). Indeed, by computer search we have found that one "minimal complexity" column permutation for this particular code is the permutation (01243567), which results in the generator matrix (cf. (3.2))

$$G(D) = \begin{pmatrix} 11111111 \\ 11110000 \\ 10101100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10110100 \\ 10001110 \end{pmatrix} D. \tag{3.4}$$

After putting the canonical generator matrix of (3.4) into "trellis-canonical" form, it becomes

$$G(D) = \begin{pmatrix} 11111111 \\ 00001111 \\ 01111111 \\ 00111111 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11111000 \\ 11111100 \\ 11111110 \end{pmatrix} D. \tag{3.5}$$

The trellis complexity of the generator matrix in (3.5) turns out to be 104 symbols per encoded bit. (This code is the seventh listed in Table VII in the Appendix.)[4]

IV. LTC versuss ACG

In this section, we will attempt to compare the trellis complexity of a number of codes to their performance. To do this, we define the "logarithmic trellis complexity" (LTC) of a code, block or convolutional, as the base-2 logarithm of the minimal trellis complexity (symbols per encoded bit), and the "asymptotic coding gain" (ACG) as the code's rate times its minimum (or free) distance. An empirical study, based on existing tables of convolutional codes ([21], [31], [22], [6], [8]), reveals the interesting fact that LTC/ACG lies between 1.5 and 2.0 for most "good" convolutional codes. For example, for the $(3,2,2,3)$ code discussed in Section III, the ratio is 1.79, and for the $(8,4,3,8)$ code, it is 1.68. By comparison, for the "NASA standard" $(2,1,6,10)$ convolutional code, for which, as for all $(n,1,m)$ convolutional codes, the minimal trellis complexity is given by the formula (1.1), the ratio is 1.60. In the Appendix, we list the (ACG, LTC) pairs for a large number of convolutional codes, and a few block codes. In Fig. 9, below, we show a scatter plot of these pairs. It is interesting to note how close most of these pairs are to the line of slope 2. This "experimental" fact may be related to a recent theorem of Lafourcade and Vardy [20], which implies that for any sequence of block codes with a fixed rate $R > 0$ and fixed value of $d/n > 0$, as $n \to \infty$

$$\liminf_{n\to\infty} \frac{\text{LTC}}{\text{ACG}} \geq 2. \tag{4.1}$$

In any case, we have been able to show in [23] that for all codes, the ratio LTC/ACG must be strictly greater than 1, a result which is similar to Theorem 3 in [19].

[4]The minimal trellis complexity of unit memory and partial unit memory convolutional codes has also been studied in [7] and [37].

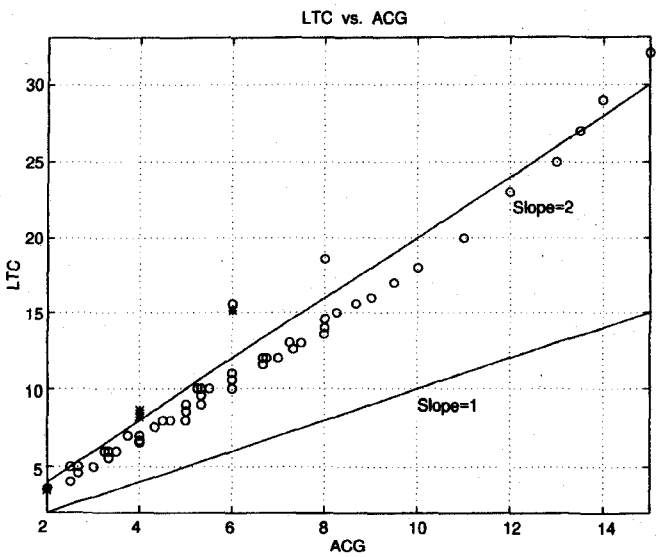

Fig. 9. A scatter plot of the pairs (ACG, LTC) for the codes listed in the Appendix. Convolutional codes are indicated with small circles and block codes with small $\times$'s.

TABLE I
BEST $(2, 1, m)$ CODES (FROM [8, pp. 85–88])

Code	LTC	ACG	LTC-ACG Ratio
(2,1,2,5)	4	2.5	1.60
(2,1,3,6)	5	3	1.67
(2,1,4,7)	6	3.5	1.71
(2,1,5,8)	7	4	1.75
(2,1,6,10)	8	5	1.60
(2,1,8,12)	10	6	1.67
(2,1,10,14)	12	7	1.71
(2,1,11,15)	13	7.5	1.73
(2,1,12,16)	14	8	1.75

TABLE II
BEST $(2, 1, m)$ CODES, CONTINUED (FROM [8, pp. 85–88])

Code	LTC	ACG	LTC–ACG Ratio
(2,1,14,18)	16	9	1.78
(2,1,15,19)	17	9.5	1.79
(2,1,16,20)	18	10	1.80
(2,1,18,22)	20	11	1.82
(2,1,21,24)	23	12	1.92
(2,1,23,26)	25	13	1.92
(2,1,25,27)	27	13.5	2.00
(2,1,27,28)	29	14	2.07
(2,1,30,30)	32	15	2.13

TABLE III
BEST $(3, 2, m)$ CODES (FROM [8, p. 90])

Code	LTC	ACG	LTC-ACG Ratio
(3,2,2,3)	3.58	2.00	1.79
(3,2,3,4)	5.00	2.67	1.87
(3,2,4,5)	6.00	3.33	1.80
(3,2,5,6)	7.00	4.00	1.75
(3,2,6,7)	8.00	4.67	1.71
(3,2,7,8)	9.00	5.33	1.69
(3,2,8,8)	10.00	5.33	1.88
(3,2,9,9)	11.00	6.00	1.83
(3,2,10,10)	12.00	6.67	1.80

V. CONCLUSION AND OPEN PROBLEMS

In this paper we have shown that every convolutional code has a unique "minimal" trellis representation, which is in many cases considerably simpler than the "conventional" trellis for the code. We have also presented a simple technique for actually constructing the minimal trellis for any convolutional code, and we have numerically computed the trellis complexity for many convolutional codes. In principle, the theory of minimal trellises for convolutional codes can be deduced from the general "Forney–Trott" theory [13], but we believe the observation that the trellis complexity of many convolutional codes, including many "nonpunctured" codes, can be thereby systematically reduced is new, as are the details of the algorithms for producing the minimal trellises.

We close with a list of research problems which suggest themselves.

TABLE IV
BEST $(4,3,m)$ CODES (FROM [8, p. 90])

Code	LTC	ACG	LTC-ACG Ratio
(4,3,3,4)	5.00	3.00	1.67
(4,3,5,5)	7.00	3.75	1.87
(4,3,6,6)	8.00	4.50	1.78
(4,3,8,7)	10.00	5.25	1.90
(4,3,9,8)	11.00	6.00	1.83

TABLE V
BEST $(3,1,m)$ CODES (FROM [8, p. 89])

Code	LTC	ACG	LTC-ACG Ratio
(3,1,2,8)	4.58	2.67	1.72
(3,1,3,10)	5.58	3.33	1.68
(3,1,4,12)	6.58	4.00	1.64
(3,1,5,13)	7.58	4.33	1.75
(3,1,6,15)	8.58	5.00	1.72
(3,1,7,16)	9.58	5.33	1.80
(3,1,8,18)	10.58	6.00	1.76
(3,1,9,20)	11.58	6.67	1.74
(3,1,10,22)	12.58	7.33	1.72
(3,1,11,24)	13.58	8.00	1.70
(3,1,12,24)	14.58	8.00	1.82
(3,1,13,26)	15.58	8.67	1.80

- A given convolutional code will, in general, have many different canonical generator matrices [24], but as we saw in Section II not all canonical generator matrices are trellis-canonical. What can be said about the class of "trellis-canonical" generator matrices?
- A theoretical explanation of the experimental observation that most of the codes shown in Fig. 9 lie near the line of slope 2 would be welcome.
- The design and implementation of Viterbi's algorithm on conventional trellises is well understood. Since the techniques described here lead to greatly reduced trellis complexity, it will be worthwhile to make a careful study of how best to implement Viterbi's algorithm on "minimal" trellises. Indeed, since the *decoding* complexity of a specific implementation of Viterbi's algorithm, and the *combinatorial* complexity of a trellis representation are related, but not identical, we regard the construction of the minimal trellis for the code as the starting point for the development of efficient algorithms, not the final answer.

TABLE VI
BEST $(4,1,m)$ CODES (FROM [8, p. 89])

Code	LTC	ACG	LTC-ACG Ratio
(4,1,2,10)	5.00	2.50	2.00
(4,1,3,13)	6.00	3.25	1.85
(4,1,4,16)	7.00	4.00	1.75
(4,1,5,18)	8.00	4.50	1.78
(4,1,6,20)	9.00	5.00	1.80
(4,1,7,22)	10.00	5.50	1.82
(4,1,8,24)	11.00	6.00	1.83
(4,1,9,27)	12.00	6.75	1.78
(4,1,10,29)	13.00	7.25	1.79
(4,1,11,32)	14.00	8.00	1.75
(4,1,12,33)	15.00	8.25	1.82
(4,1,13,36)	16.00	9.00	1.78

- From our current viewpoint, punctured convolutional codes are just codes whose trellis module has fewer edge symbols than would normally be expected. This is because in the scalar matrix $\tilde{G}$ for a punctured code, certain entries are guaranteed to be zero. For example, for a $(4,3,3)$ punctured code, the matrix $\tilde{G}$ has the "template" structure

$$\tilde{G} = \begin{pmatrix} x & x & x & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & x & x & 0 \\ 0 & 0 & 0 & x & x & x & x & x \end{pmatrix}$$

where the x's can be arbitrary (actually there are restrictions on the x's which depend in detail on how the code is constructed), but the eight zero positions must be respected. Any $(4,3,3)$ convolutional code with such a "template" structure will have trellis complexity at most $4/3 \cdot 2^4 = 21.3$. An obvious question is whether other

TABLE VII
SOME BLOCK CODES AND PARTIAL UNIT MEMORY CONVOLUTIONAL CODES

Code	LTC	ACG	LTC-ACG Ratio
[8,4,4] *Self-dual Code*	3.46	2.00	1.73
[24,12,8] *Golay Code*	8.22	4.00	2.06
[32,16,8] *BCH Code*	8.64	4.00	2.16
[48,24,12] *Self-dual Code*	15.13	6.00	2.52
[n,$n-1$,2] *Parity-check Code*	2.00	$\frac{2(n-1)}{n}$	$\frac{n}{n-1}$
[n,1,n] *Repetition Code*	$1+\log_2 n$	1	$1+\log_2 n$
(8,4,3,8) *PUM Code*	6.70	4.00	1.68
(24,12,7,12) *PUM Code*	15.58	6.00	2.60
(24,12,10,16) *PUM Code*	18.58	8.00	2.32

"low-complexity templates" support good convolutional codes. (A similar observation about code "templates" is made in [4].)

- In our computer-aided search for the "best" column permutation of the $(8,4,3,8)$ code, we found that each of the $8! = 40\,326$ possible column permutations had minimal trellis complexity either 120 or 104. This strongly suggests an equivalence among permutations, which if understood theoretically, could make it much simpler to find the best column permutation.

Finally, we remark that when the bulk of this paper was written, we were not aware of the important earlier work of Sidorenko and Zyablov [32], which deals explicitly with the minimal trellis for a convolutional code, and we wish to acknowledge their priority. Their work, like ours, develops the theory of minimal trellises for convolutional codes from the corresponding theory for block codes. However, their trellis construction is based on the parity-check matrix of the code rather than the generator matrix, and their emphasis is quite different. One advantage of the Sidorenko–Zyablov approach is that it leads to the following upper bound on the number of nodes at depth i in the minimal trellis for a (n,k,m) convolutional code [32, Theorem 1]

$$N_i \leq 2^{m+\min(k,n-k)}.$$

It is not easy to derive this bound using our methods. On the other hand, the present paper contains a number of things not present in [32], among them being

- The observation that the minimal trellis for a punctured convolutional code is at least as simple as the "punctured" trellis.
- The concept of a "trellis-canonical" generator matrix for a convolutional code, and an algorithm for computing one.
- The ACG versus LTC comparison for block and convolutional codes.

(Even more recently, Sidorenko, Markarian, and Honary [33] have discussed the construction of minimal trellises for some convolutional codes using the Shannon product of "elementary" trellises.)

APPENDIX
TABLES OF LTC VERSUS ACG

In this appendix, we list the "ACG" and the "LTC" for a large number of "good" convolutional codes, and a few block codes (see Tables I–VII). A scatter plot of these (ACG, LTC) pairs appears as Fig. 9 in Section IV.

REFERENCES

[1] K. Abdel-Ghaffar, R. J. McEliece, and G. Solomon, "Some partial unit memory convolutional codes," *JPL TDA Progress Rep.*, vol. 42–107, pp. 57–72, Nov. 1991. Also in *Proc. 1991 Int. Symp. on Information Theory*, p. 196.

[2] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
[3] G. Bégin and D. Haccoun, "High-rate punctured convolutional codes: Structure properties and construction technique," *IEEE Trans. Commun.*, vol. 37, pp. 1381–1385, Dec. 1989.
[4] I. Bocharova and B. Kudryashov, "Nonsyndrome maximum likelihood decoding of linear codes using a trellis," in *Proc. 4th Int. Workshop on Algebraic and Combinatorial Coding Theory* (Novogrod, Russia, Sept. 11–17, 1994), pp. 35–39.
[5] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 1, pp. 97–100, Jan. 1979.
[6] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New short constraint length convolutional code construction for selected rational rates," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 5, pp. 794–800, Sept. 1982.
[7] U. Dettmar and U. Sorger, "On maximum likelihood decoding of unit memory codes," in *Proc. 6th Swedish–Russian Int. Workshop on Information Theory*, Aug. 1993, pp. 184–188.
[8] A. Dholakia, *Introduction to Convolutional Codes with Applications*. Boston, MA: Kluwer, 1994.
[9] S. Dolinar, L. Ekroot, A. Kiely, R. McEliece, and W. Lin, "The permutation trellis complexity of linear block codes," in *Proc. 32nd Annual Allerton Conf. on Communication, Control, and Computing* (Allerton Park, IL, Sept. 1994), pp. 60–74.
[10] G. D. Forney, Jr., "Convolutional codes I: Algebraic structure," *IEEE Trans. Inform. Theory*, vol. IT-16, no. 6, pp. 268–278, Nov. 1970.
[11] ——, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–276, Mar. 1973.
[12] ——, "Coset codes—Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, no. 5, pp. 1152–1187, Sept. 1988.
[13] G. D. Forney, Jr. and M. D. Trott, "The dynamics of group codes: State spaces, trellis diagrams, and canonical encoders," *IEEE Trans. Inform. Theory*, vol. 39, no. 5, pp. 1491–1513, Sept. 1993.
[14] G. D. Forney, R. Johannesson, and Z.-X. Wan, "Minimal and canonical rational generator matrices for convolutional codes," this issue, pp. 1865–1880.
[15] D. Haccoun and G. Bégin, "High-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Trans. Commun.*, vol. 37, pp. 1113–1125, Nov. 1989.
[16] B. Honary, G. Markarian, and M. Darnell, "Trellis decoding for block codes," in *Proc. 3rd IEE Int. Symp. on Communication Theory and Applications* (Ambleside, UK, July 1993), pp. 79–93.
[17] A. Kiely, S. Dolinar, R. McEliece, L. Ekroot, and W. Lin, "Trellis decoding complexity of linear block codes," this issue, pp. 1687–1697.
[18] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory*, vol. 41, no. 6, pp. 1924–1937, Nov. 1995.
[19] A. Lafourcade and A. Vardy, "Asymptotically good codes have infinite trellis complexity," *IEEE Trans. Inform. Theory*, vol. 41, no. 2, pp. 555–559, Mar. 1995.
[20] ——, "Lower bounds on trellis complexity of block codes," *IEEE Trans. Inform. Theory*, vol. 41, no. 6, pp. 1938–1954, Nov. 1995.
[21] K. Larsen, "Short convolutional codes with maximal free distance for rates $1/2, 1/3$, and $1/4$, *IEEE Trans. Inform. Theory*, vol. IT-19, no. 2, pp. 371–372, May 1973.
[22] G. S. Lauer, "Some optimal partial-unit-memory codes," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 2, pp. 540–547, Mar. 1979.
[23] W. Lin and R. J. McEliece, "Asymptotic coding gain and trellis complexity," in *Proc. 33rd Allerton Conf. on Communication, Control, and Computing* (Allerton Park, IL, Oct. 4–6, 1995), pp. 313–322.
[24] K. Lumbard and R. J. McEliece, "Counting minimal generator matrices," in *Proc. 1994 IEEE Inter. Symp. on Information Theory* (Trondheim, Norway, June 1994), p. 18.
[25] J. L. Massey, "Foundations and methods of channel coding," in *Proc. Int. Conf. on Information Theory and Systems (NTG-Fachberichte)*, vol. 65, pp. 148–157, 1978.
[26] R. J. McEliece, *The Theory of Information and Coding*. Reading, MA: Addison-Wesley, 1977.
[27] ——, "The Viterbi decoding complexity of linear block codes," in *Proc. 1994 IEEE Int. Symp. on Information Theory* (Trondheim, Norway, June 1994,) p. 341.
[28] ——, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 4, pp. 1072–1092, July 1996.
[29] ——, "The algebraic theory of convolutional codes," chapter in the *Handbook of Coding Theory*, in preparation.
[30] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, vol. 34, no. 5, pp. 1049–1053, Sept. 1988.
[31] E. Paaske, "Short binary convolutional codes with maximal free distance," *IEEE Trans. Inform. Theory*, vol. IT-20, no. 5, pp. 683–688, Sept. 1974.
[32] V. Sidorenko and V. Zyablov, "Decoding of convolutional codes using a syndrome trellis," *IEEE Trans. Inform. Theory*, vol. 40, no. 5, pp. 1663–1666, Sept. 1994.
[33] V. Siderenko, G. Markarian, and B. Honary, "Minimal trellis design for linear codes based on the Shannon product," this issue, pp. 2048–2053.
[34] A. Vardy and F. R. Kschischang, "Proof of a conjecture of McEliece regarding the expansion index of the minimal trellis," *IEEE Trans. Inform. Theory*, this issue, pp. 2027–2034.
[35] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.
[36] J. K. Wolf, "Efficient maximum likelihood decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-24, no. 1, pp. 76–80, Jan. 1978.
[37] V. Zyablov and V. Sidorenko, "Soft decision maximum likelihood decoding of partial unit memory codes," *Probl. Inform. Transm.*, vol. 28, no. 1, pp. 18–22, Jan.–Mar. 1992.